Standard Instrument Control Library

HP-IB User's Guide for Windows

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

Notice

The information contained in this document is subject to change without notice.

Hewlett-Packard Company (HP) shall not be liable for any errors contained in this document. HP makes no warranties of any kind with regard to this document, whether express or implied. HP specifically disclaims the implied warranties of merchantability and fitness for a particular purpose. HP shall not be liable for any direct, indirect, special, incidental, or consequential damages, whether based on contract, tort, or any other legal theory, in connection with the furnishing of this document or the use of the information in this document.

Warranty Information.

A copy of the specific warranty terms applicable to your Hewlett-Packard product and replacement parts can be obtained from your local Sales and Service Office.

Restricted Rights Legend.

Use, duplication or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause of DFARS 252.227-7013.

Use of this manual and magnetic media supplied for this product are restricted. Additional copies of the software can be made for security and backup purposes only. Resale of the software in its present form or with alterations is expressly prohibited.

Microsoft and QuickC are registered trademarks, and VisualC++, Windows and Windows NT are trademarks of the Microsoft Corporation.

Borland and Turbo C are registered trademarks of Borland International, Inc.

Printing History

This is the first edition of the *Standard Instrument Control Library HP-IB User's Guide* for Windows platforms.

December 1993 - First Edition

Contents

1

Introduction

Introduction

In This Manual

Welcome to the Standard Instrument Control Library *HP-IB User's Guide*. This manual will guide you through hardware and software installation and configuration of HP-IB interfaces supported with the Standard Instrument Control Library (abbreviated SICL for purposes of this manual) on Microsoft Windows and Windows NT platforms. The manual also contains a getting started section and examples to help you build a SICL application with the popular Microsoft and Borland C/C++ language tools and some considerations to remember when designing for Windows. The appendix contains a short section on HP-IB bus theory and connectivity, and a section on tips for users moving from the HP-IB Command Library products to the Standard Instrument Control Library.

This manual should be used in conjunction with the *Standard Instrument Control Library Reference Manual* which contains a complete description of features and use of the library.

Related Documents

Standard Instrument Control Library Reference Manual. Contains a complete description of the Standard Instrument Control Library with a task reference introduction to the various features of the library, some information on specific interfaces, and a full language reference of all library functions.

Standard Instrument Control Library Quick Reference Guide. Handy reference to all library functions, error codes, commonly used constants and formatted I/O strings.

Standard Instrument Control Library Serial User's Guide for Windows. Describes the use of SICL for instrument control over standard serial (e.g. RS-232) ports. Included with SICL/Serial product.

2

Preparing to Install SICL

Preparing to Install SICL

This chapter provides information for preparing your system to use the Standard Instrument Control Library. The main sections of this chapter are as follows:

- Verifying the Product Package
- System Software Requirements
- System Hardware Requirements

Verifying the Product Package

You should have received the following items as part of your HP-IB for Windows order:

- HP SICL manual set

  Standard Instrument Control Library HP-IB User's Guide for Windows.
  Standard Instrument Control Library Reference Manual
  Standard Instrument Control Library Quick Reference Guide

- Installation media

  This SICL software is supplied on 3.5 inch floppies, one floppy for Windows 3.1 and one for Windows NT (if ordered). If you require 5.25 inch floppies, contact the number listed in the Start-up Assistance information

- HP Software Product License Agreement
- HP-IB interface card (as part of the HP 82341A product)
- Computer Plug-in Accessories Warranty and Support information
- Complimentary Start-up Assistance information
- Customer Registration/Questionnaire Card

If you also ordered HP-IB cables, these may be shipped separately.

HP SICL for Windows 3.1

In order to use SICL/HP-IB on your Windows 3.1 system, you must have the following software:

- MS-DOS 5.0 or later
- Microsoft Windows (or Windows for Workgroups) 3.1 or later
- C/C++ Language tools
  Microsoft C 6.0 or later, Visual C 1.0 or later, QuickC for Windows 1.0 or later, or other equivalent Microsoft C language tools
  Borland C and Turbo C 3.0 or later.
  Other Microsoft/Borland compatible C language tools

NOTE

When installing your Windows 3.1 language tools, make sure that the libraries for programming with the large memory model are loaded. If not, you may have difficulty developing your SICL application.

HP SICL for Windows NT

If you have ordered SICL/HP-IB for Windows NT, you must have the following software:

- Microsoft Windows NT 3.1 or later
- C/C++ Language tools
  Microsoft 32-bit Visual C 1.0 or later or other equivalent Microsoft C language tools for Windows NT
  Other Windows NT compatible C language tools

System Hardware Requirements

The Standard Instrument Control Library has the following minimum hardware system requirements:

- 80386-based personal computer
- Sufficient memory to run Windows 3.1 (2-4 MBytes RAM) or Windows NT (16 MByte RAM). Additional RAM may improve overall system performance.
- Available ISA or EISA I/O slots for plug-in HP-IB interface card.
- Available hard disk space for the SICL software (refer to the setup program for specific requirements).

3

Installing Hardware

Installing Hardware

This chapter covers the installation procedures for the HP 82341 High Performance HP-IB interface and the HP 82335 HP-IB interface. Refer to the section for the interface you are using. Both interfaces are supported on Windows 3.1, but only the HP 82341 is supported on Windows NT.

It is strongly recommended that you install the HP-IB interface in your computer before you install the SICL software.

NOTE

You cannot place two HP-IB interface boards in adjacent slots in a personal computer due to the HP-IB cable connector width. You must space them apart by at least one slot.

Depending on your computer, you also may be unable to use the end slots if the computer housing does not provide adequate space to connect the HP-IB cable.

Unpacking the Interface Card

CAUTION

Observe the following precautions to reduce the risk of damaging the interface board.

- Protect the interface board from static electricity. The interface can be damaged by static electricity. For protection, the interface is packed in an antistatic bag. Leave the interface board in its antistatic bag until you're ready to install it. Save the antistatic bag so you can protect the interface if you have to remove it from the computer.
- Handle the interface board gently. Do not drop it or handle it roughly. Be careful while unpacking and installing the interface board.
- Hold the interface only by its edges. Never touch any other part of the interface including the ISA connector tab.

Setting the Configuration Switches

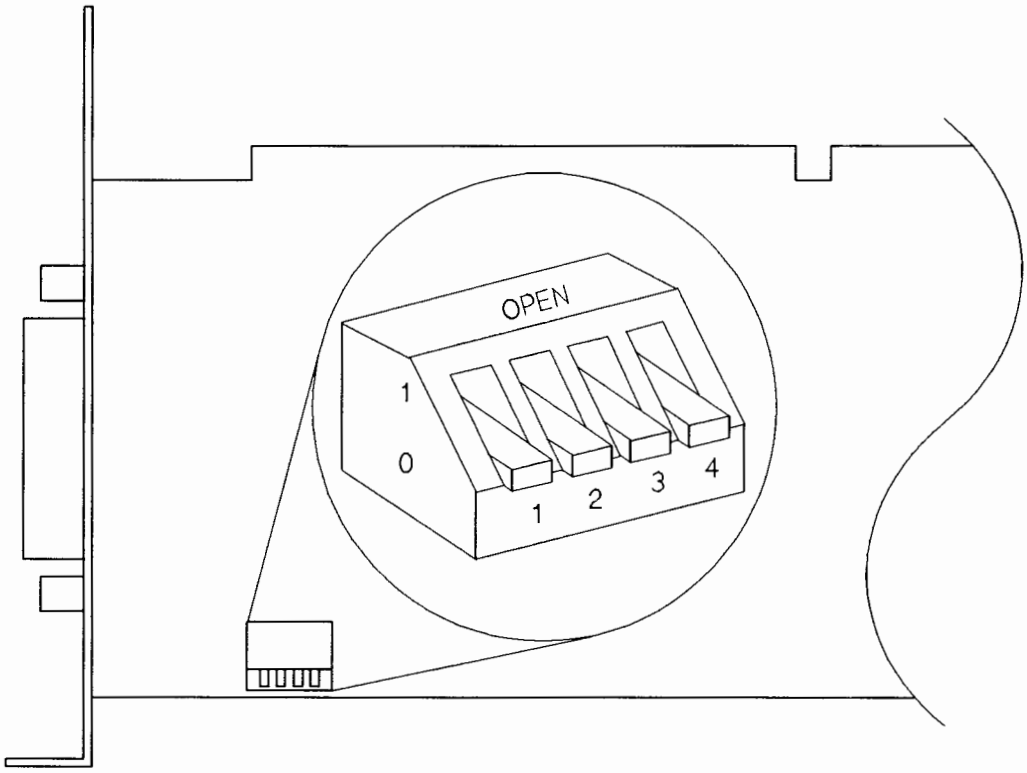
The following sections explain switch settings. Please refer to the section for the interface you are installing.

Setting Switches on the HP 82341

NOTE

The HP 82341 interface card has two empty sockets. These sockets are used for an upgrade that is not supported on the personal computer. The card will only function properly with these sockets empty.

The configuration switches on the HP 82341 interface set the interface's I/O (port) base address. They're set at the factory as shown in the following figure which selects the default hexadecimal I/O base address of 250.



In most cases, you will not need to change the default switch setting. However, if you are installing more than one HP 82341 interface, you will need to change the switch settings on the additional interfaces so that each one has a unique switch setting. Use the following table of switch positions to select the desired address ranges for each interface.

NOTE

The selected I/O address ranges must not conflict with other I/O interfaces installed in your computer, including other manufacturer's products (e.g. LAN interfaces, etc.). Refer to the documentation for the other interfaces and the table below to select unique addresses for all interfaces in your computer.

Because the HP 82341 interface is I/O port mapped and not memory mapped, it is not necessary to exclude address ranges for memory management software on your computer.

HP 82341 Switch Settings

Switches 1 2 3 4	I/O Base Address (Hexadecimal)	I/O Address Range Used (Hexadecimal)
0 0 0 0	250	250-257
1 0 0 0	270	270-277
0 1 0 0	350	350-357
1 1 0 0	370	370-377
0 0 1 0	220	220-227
1 0 1 0	280	280-287
0 1 1 0	390	390-397

All other hardware settings for the HP 82341 are configured by the software. Continue on to "Installing and Configuring Software" to complete the SICL installation process.

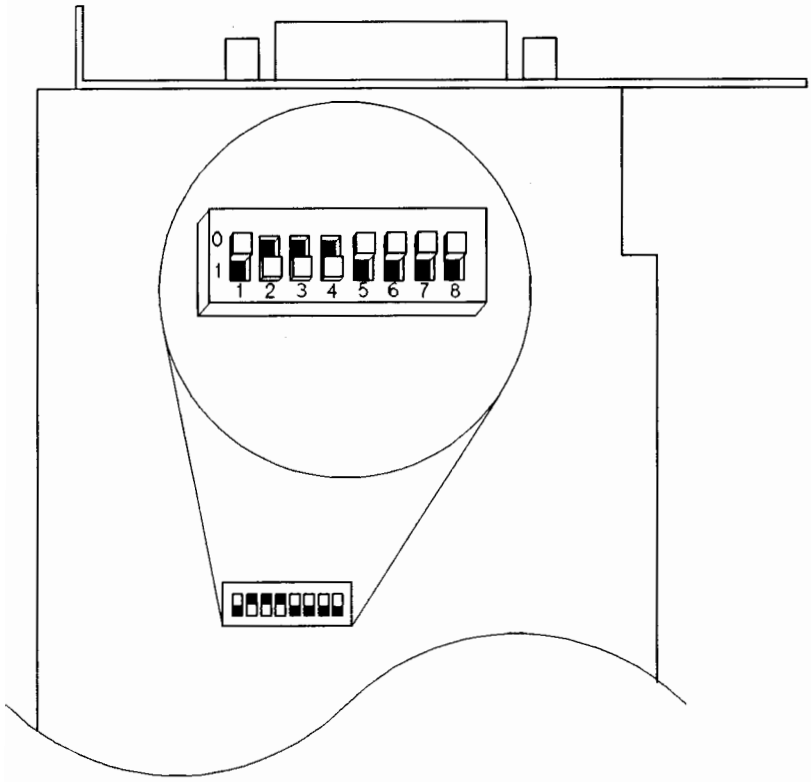
Setting Switches on the HP 82335

NOTE

The HP 82335 interface is only supported on Windows 3.1. It cannot be used on Windows NT.

Additionally, SICL does not support interrupt, SRQ or non-controller functions with the HP 82335. If you require this functionality, you must use the HP 82341 interface hardware.

The configuration switches on the HP 82335 interface set the interface's operating parameters. They're set at the factory as shown in the following figure which specifies memory segment DC00. Switches 1 through 4 determine the memory address of the interface. Switches 5 through 8 are not used by SICL.



In most cases, you will not need to change the default switch setting. However, if you are installing more than one HP 82335 interface, you will need to change the switch settings on the additional interfaces so that each one has a unique switch setting. Use the following table of switch positions to select the desired address ranges for each interface.

HP 82335 Switch Settings

Switches 1 2 3 4 5 6 7 8	Memory Segment Range
0 1 1 1 0 0 0 0	DC00-DFFF
0 1 1 0 0 0 0 0	D800-DBFF
0 1 0 1 0 0 0 0	D400-D7FF
0 1 0 0 0 0 0 0	D000-D3FF
0 0 1 1 0 0 0 0	CC00-CFFF

NOTE

The chosen memory address ranges must not conflict with other interfaces installed in your computer, including other manufacturer's products. Refer to the documentation for the other interfaces and the table above to select unique memory addresses for all interfaces in your computer.

Additionally, these memory ranges may be used by memory management software including Windows 3.1. You must specifically exclude the use of the segment required by the HP 82335 from those segments available to the memory management software.

After installing the HP 82335 and booting the computer, check for a memory manager device line (for example, `DEVICE=EMM386.EXE`) in the `CONFIG.SYS` file in the root directory. Add a parameter to exclude the memory segment (for example, `X=DC00-DCFF`) to the memory manager line for the segment used by the HP 82335 interface. Remember, you must reboot the computer for any changes to take effect.

Before you start Windows, you must also edit the `SYSTEM.INI` file in your Windows directory and add a memory exclude line to the `[386enh]` section (for example, `EMMEXCLUDE=0DC00-0DFFF`).

Use the following steps to install the HP-IB interface in your computer:

1. Power down the computer and all its peripherals and disconnect the power cord from the computer.
2. Remove the necessary covers to expose the ISA/EISA expansion slots and remove the backpanel cover plate for the desired slot as described in your computer documentation.

NOTE

You cannot place two HP-IB interface boards in adjacent slots due to the HP-IB cable connector width. You must space them apart by at least one slot.

Depending on your computer, you also may not be able to use the end slots if the computer housing does not provide adequate space to connect the HP-IB cable.

3. Insert the HP-IB interface edge connector into the expansion slot connector of the computer. Make sure the interface is fully seated by pushing firmly on the edge of the board with the palm of your hand. The HP-IB connector should extend through the backpanel opening to allow cable installation.
4. Replace the backpanel cover plate screw to hold the interface in place. (Save the blank cover plate for use if the interface is later removed.)
5. Replace the covers removed in step 2 as described in your computer documentation.

6. Connect an HP-IB cable to the interface using one of the following cables:

   - HP 10833A (1 meter).
   - HP 10833B (2 meters).
   - HP 10833C (4 meters).
   - HP 10833D (0.5 meter).
   - HP 8120-3448 (6 meters).
   - HP 8120-3449 (8 meters).
   - Other cables designed for use with IEEE-488 HP-IB/GPIB buses

   Tighten the HP-IB connector screws finger tight only.

7. Re-connect the power cord and power up the computer and peripherals.

Continue on to "Installing and Configuring Software" to complete the SICL installation process.

4

Installing and Configuring Software

Installing and Configuring Software

This chapter describes the software installation procedure as well as the configuration process for setting up the interfaces you will be using.

Installing the SICL Software

Refer to the appropriate installation section below for the Windows environment you are using.

If you have ordered SICL software for both Windows 3.1 and Windows NT, you must perform the installation steps as described for each version.

Installation for Windows 3.1

Follow these steps to load the SICL software:

1. If you have not already done so, start up Windows by typing **`win`** at the command prompt.
2. Locate the SICL media labeled for Windows 3.1 and insert the disk in the appropriate floppy drive. We will use drive **`a:`** in these instructions.
3. From the Program Manager menu choose **`File | Run`**.
4. Type **`a:setup`** and carriage return.
5. The setup program prompts you for the directory in which to install the files. The default is **`\SICL`** on your primary hard drive. Click the **`Install`** button to use the default, or click **`Set Location`** to type in the desired directory and then click **`Continue`**.
6. The setup program automatically copies all of the files and prompts you when it has completed successfully.
7. When setup is complete, remove the SICL distribution disk and store it in a safe location.

The setup program creates an HP SICL program group and defines icons for a readme file, the **`I/O Config`** configuration utility, the **`SICL Message Viewer`** utility, and **`SICL Help`**.

The Windows **`WIN.INI`** file is also modified by the setup program in order to record the location of the SICL directory. This change will have no other effect on your Windows environment.

> **NOTE**
>
> It is very important that you look through the readme file before proceeding with any application development. Double click on the **`Read Me First`** icon in the HP SICL group to view the file.

Installation for Windows NT

> **NOTE**
>
> You must have system administrator privileges in order to install SICL on Windows NT.

Follow these steps to load the SICL software:

1. Locate the SICL media labeled for Windows NT and insert the disk in the appropriate floppy drive. We will use drive a: in these instructions.
2. From the Program Manager menu choose **`File | Run`**.
3. Type **`a:setup`** and carriage return.
4. The setup program prompts you for the directory in which to install the files. The default is **`\SICL`** on your primary hard drive. Click the **`Install`** button to use the default, or click **`Set Location`** to type in the desired directory and then click **`Continue`**.
5. The setup program automatically copies all of the files and prompts you when it has completed successfully.

6. When setup is complete, remove the SICL distribution floppy and store in a safe location.

The setup program creates an HP SICL program group and defines icons for a **`Windows NT Read Me First`** file, the **`Windows NT I/O Config`** utility, and **`SICL Windows NT Help`**.

If you have also installed SICL for Windows 3.1, be sure to run the Windows 3.1 SICL programs only when running Windows 3.1, and the Windows NT SICL programs (so noted) when running Windows NT.

NOTE

It is very important that you look through the readme file before proceeding with any application development. Double click on the **`Windows NT Read Me First`** icon in the HP SICL group to view the file.

Some newer personal computers have 32-bit EISA backplanes. If you have installed your HP 82341 HP-IB interface in an EISA slot, you may wish to run the EISA configuration utility provided with your computer in order to properly assign hardware resources to the interface and avoid system conflicts.

When you ran the SICL setup program, the EISA "`.CFG`" file was placed in the `EISACFG` directory under the SICL base directory (e.g. `C:\SICL\EISACFG` if you installed the software in the default location). The "`.CFG`" file is also available directly from the SICL installation disk if this is more convenient.

When you run the EISA configuration utility and assign resources, be sure to note the settings for use with the SICL `I/O Config` utility described in the next section.

Configuring the HP-IB interfaces

NOTE

On Windows 3.1, if there are any SICL applications running when changes are made using the **I/O Config** utility, these changes will not take affect until all currently executing SICL applications have completed.

If you are running Windows NT, you must have system administrator privileges in order to run the **I/O Config** utility. You will also be instructed to reboot the computer if you add or modify interfaces with the **I/O Config** utility. This is required in order to load or update the driver files in the Windows NT kernel.

To complete installation of SICL and the hardware interfaces on Windows 3.1 or Windows NT, execute the **I/O Config** utility by double clicking on the **I/O Config** icon in the HP SICL group. If you have not yet installed the hardware interfaces in your computer, you should do so before proceeding.

The **I/O Config** utility is an interactive program that searches your system for installed interfaces that are supported by SICL. You must select the interfaces you wish to use and the **I/O Config** utility will choose a number of default parameters needed to configure the interface. Any configurable hardware options, such as which computer interrupt line to use, plus important SICL configuration details will be displayed for the selected interface. In most cases you will be able use the automatic defaults. If you need to change something, just click on the field and type in the new value.

The name of the interface, such as **hpib7**, and the logical unit number should be noted for later use in your application. The configuration utility can be run at any time to check or make changes to the settings of configured interfaces.

If you have any questions about the use of the **I/O Config** utility for a specific interface, select **Help** from the **I/O Config** utility menu.

When you have finished configuring the interfaces, click on **OK** and **Yes** to store the changes you have made. You are now ready to begin creating

your application. The next chapter includes important information about programming with SICL plus some helpful examples to get you started.

5

Getting Started using SICL

Getting Started using SICL

This chapter will help you to get started programming with SICL. The first section will step through a simple example to allow you to verify your configuration and introduce you to some of the basic features provided by SICL.

The second section contains important details about creating applications using SICL in the Windows environment. If you have not programmed with SICL before, you may first wish to look through the task reference section of the SICL *Reference Manual* to familiarize yourself with the capabilities of SICL. The *Reference Manual* is available as on-line help by double clicking on the **Help** icon in the SICL program group.

The last section of this chapter presents a more detailed Windows application example with additional SICL features and illustrates many of the important Windows programming considerations discussed in the second section.

IDN Example

In this section we will build and run a simple example program that queries an HP-IB instrument for its identification string. This example uses the QuickWin or EasyWin feature of Microsoft and Borland compilers on Windows 3.1, or builds a console application for Windows NT. The example will show you some of the basic functions of SICL and will help you verify your configuration. An overview of the program immediately follows the procedure for building the program.

Building and Running the Program

The IDN example files are located in the `C\SAMPLES\IDN` directory under the SICL base directory (e.g. `C:\SICL\C\SAMPLES\IDN` if you installed SICL in the default location). The directory contains the source program and a number of files to help you build the example with specific compilers. Here is a summary of the files provided:

`IDN.C`	Windows 3.1 and Windows NT Example program source file.
`IDN.DEF`	Module definition file for the IDN example program.
`MSCIDN.MAK`	Windows 3.1 makefile for Microsoft C and Microsoft SDK compilers.
`VCIDN.MAK`	Windows 3.1 project file for Microsoft Visual C++.
`QCIDN.MAK`	Windows 3.1 project file for Microsoft QuickC for Windows.
`BCIDN.MAK`	Windows 3.1 makefile for Borland C command line compilation.
`BCIDN.PRJ`	Windows 3.1 project file for Borland C Integrated Development Environment.

If you have purchased and installed SICL for Windows NT, the following files will also be available:

`MSCIDNNT.MAK`	Windows NT Software Development Kit compilers.
`VCIDNNT.MAK`	Windows NT project file for Microsoft Visual C++.

Here is a list of the steps you should follow to build and run the IDN example (see "SICL Programming Considerations" later in this chapter for details on creating your own makefile or project):

1. Connect an instrument to your HP 82341 or 82335 HP-IB interface that is compatible with Standard Commands for Programmable Instruments (SCPI) directives.
2. Change directories to the location of the example (e.g. `CD \SICL\C\SAMPLES\IDN`).
3. The program assumes that the HP-IB interface name is `hpib7` (set using `I/O Config`) and that the instrument is at bus address `0`. If necessary, use an editor to modify the interface name and instrument address on the `DEVICE_ADDRESS` definition line in the `IDN.C` source file.
4. If you use the command line interface for your compiler:
   a. Compile the program using the makefile from the command prompt. For Borland compilers: type `make bcidn.mak`. For Microsoft compilers: type `nmake mscidn.mak` (or `nmake mscidnnt.mak` for Windows NT).
   b. Execute the program. Select `File | Run` from the Windows Program Manager menu (e.g., type `C:\SICL\C\SAMPLES\IDN\IDN` in the input box. Click on `OK`. On Windows NT, you should execute the program from a Windows NT console command prompt.
5. If you use the Windows interface for your compiler:
   a. Select and load the appropriate project or makefile. Use `Project | Open Project` for Borland, or `Project | Open` for Microsoft compilers.
   b. For Microsoft compilers, set the include file path by selecting `Options | Directories` and in the `Include File Path` box, add a semicolon followed by `C:\SICL\C` (if you installed SICL in the default location). This is supplied in the project file for Borland compilers.
   c. Compile the program using `Compile | Build All` for Borland, or `Project | Re-build All` for Microsoft compilers.

d. To execute the program, select **Run | Run** for Borland, or **Project | Execute** or **Run | Go** for Microsoft compilers. On Windows NT, you should execute the program from a Windows NT console command prompt.

If the program runs correctly, the following is an example of the output:

```
HEWLETT-PACKARD,54601A,0,1.7
```

If the program does not run, refer to the chapter on "Troubleshooting" for assistance in correcting the problem.

Program Overview

The source file **IDN.C** is listed here:

```
////////////////////////////////////////////////////////////////////////
//
// The following simple demonstration program uses the Standard
// Instrument Control Library to query an HP-IB instrument for
// an identification string and then prints the result.
//
// Edit the DEVICE_ADDRESS line below to specify the address of the
// device you want to talk to.  For example:
//
//     hpib7,0    - refers to an HP-IB device at bus address 0
//                  connected to an interface named "hpib7" by the
//                  I/O Config utility.
//
//     hpib7,9,0  - refers to an HP-IB device at bus address 9,
//                  secondary address 0, connected to an interface
//                  named "hpib7" by the I/O Config utility.
//
// Note that this program is meant to be built as a Windows 3.1
// QuickWin or EasyWin program, or Windows NT console application.
// For Windows 3.1, it must be compiled with the Large memory model!
//
////////////////////////////////////////////////////////////////////////
```

```
#include <stdio.h>  // for printf()
#include "sicl.h"  // Standard Instrument Control Library routines

#define DEVICE_ADDRESS "hpib7,0"   // Modify this line to match your setup

void main(void)
{
   INST id;                  // device session id
   char buf[256] = { 0 };    // read buffer for idn string

   #if defined(__BORLANDC__)
     _InitEasyWin();  // required for Borland EasyWin programs.
   #endif

   // Install a default SICL error handler that logs an error message and
   // exits.  On Windows 3.1, view messages with the SICL Message Viewer,
   // and on Windows NT use the Windows NT Event Viewer.
   ionerror(I_ERROR_EXIT);

   // Open a device session using the DEVICE_ADDRESS
   id = iopen(DEVICE_ADDRESS);

   // Set the I/O timeout value for this session to 1 second
   itimeout(id, 1000);

   // Write the *RST string (and send an EOI indicator) to put the instrument
   // in a known state.
   iprintf(id, "*RST\n");

   // Write the *IDN? string and send an EOI indicator, then read
   // the response into buf.
   // For Windows 3.1, this will only work with the Large memory model
   // since ipromptf expects to receive far pointers to the format strings.

   ipromptf(id, "*IDN?\n", "%t", buf);
   printf("%s\n", buf);

   iclose(id);

   // For Windows 3.1, call _siclcleanup before exiting to release
   // resources allocated by SICL for this application.  This call is a
   // no-op for Windows NT.
   _siclcleanup();
}
```

sicl.h

The **sicl.h** file is included at the beginning of the file to provide the function prototypes and constants defined by SICL.

INST

Notice the declaration of **INST id** at the beginning of **main**. The type **INST** is defined by SICL and is used to represent a unique identifier that will describe the specific device or interface that you are communicating with. The **id** is set by the return value of the SICL **iopen** call, and will be set to 0 if **iopen** fails for any reason.

ionerror

The first SICL call, **ionerror**, installs a default error handling routine that is automatically called if any of the subsequent SICL calls result in an error. **I_ERROR_EXIT** specifies a built-in error handler that will print out a message about the error and then exit the program. If desired, a custom error handling routine could be specified instead. On Windows 3.1, the error messages may be viewed by executing the SICL **Message Viewer** utility in the SICL Windows Group. For Windows NT, these messages may be viewed with the Windows NT **Event Viewer**.

iopen

Then an **iopen** call is made. The parameter string "hpib7,0" passed to **iopen** specifies the HP-IB interface, followed by the bus address of the instrument. The interface name, "hpib7", is the name given to the interface during execution of the SICL **I/O Config** utility. The bus (primary) address of the instrument follows, in this case "0", and is typically set with switches on the instrument, or from the front panel of the instrument.

NOTE

If you are connecting to a VXI card cage through an HP E1405/6 Command Module or equivalent, the primary address passed to **iopen** corresponds to the address of the Command Module, and then a secondary address must be specified to select a specific instrument in the card cage. Secondary addresses of 0, 1, 2, ... 31, correspond to VXI instruments at logical addresses of 0, 8, 16, ... 248, respectively. For example, the **iopen** address string "hpib7,9,0" would specify the VXI instrument at logical address 0 (usually the Command Module), accessed through a Command Module at address 9 on the HP-IB bus.

You may wish to modify the program to set the interface name and instrument address to those applicable for your setup. Refer to the section on addressing in the SICL *Reference Manual* for a complete description of how to use SICL's addressing capabilities.

The **iopen** call creates a SICL device session, for communication to this specific device (or instrument), with a unique identifier returned. This identifier is used with subsequent SICL calls to indicate which device is being addressed. If only the interface name had been passed to **iopen**, this would have created an "interface session" instead. In general, device session programming is recommended over interface session programming. This keeps your program from having to understand and manage the intrinsics of the HP-IB bus.

itimeout

Next, **itimeout** is called to set the length of time that SICL will wait for an instrument to respond. The specified value will depend on the needs of your configuration. Different timeout values can be set for different sessions as needed.

iprintf and **ipromptf**

SICL provides formatted I/O functions that are patterned after those used in the C programming language. These SICL functions support the standard ANSI C format strings, plus additional formats defined specifically for instrument I/O. The SICL **iprintf** call sends the SCPI "***RST**" command to the instrument that puts it in a known state. Then **ipromptf** is used to query the instrument for its identification string. The string is read back into **buf** and then printed to the screen. (Separate **iprintf** and **iscanf** calls could have been used to perform this operation.) The **%t** read format string specifies that an ASCII string is to be read back, with end indicator termination. SICL automatically handles all addressing and HP-IB bus management necessary to perform these reads and writes to instrument.

iclose and **_siclcleanup**

The **iclose** function closes the device session to this instrument (**id** is no longer valid after this point). And finally, a call to **_siclcleanup** tells Windows 3.1 that the program is done and that the SICL I/O resources are no longer needed. Windows NT does not require the **_siclcleanup** call.

Refer to the HP SICL *Reference Manual* or SICL **Help** for more detailed information on these SICL calls and to learn about all of the functions provided by SICL.

SICL Programming Considerations for Windows

Be sure to read the appropriate section below before developing SICL applications to run in the Windows 3.1 or Windows NT environment.

The information in this section is meant to be used in conjunction with the complete description of the features of SICL found in the SICL *Reference Manual*.

SICL and Windows 3.1

The SICL Header File

The SICL header file `sicl.h` should be included at the beginning of each file containing functions that call SICL. The header file contains function prototypes for all SICL functions plus definitions for all SICL constants and error codes. To include the SICL header, simply add the following line to your source file:

```
#include "sicl.h"
```

The SICL header file is located in the "C" directory under the SICL base directory (e.g. `C:\SICL\C` if you installed SICL in the default location). You may wish to add this directory path to the include file path used by your C programming language tools.

Interrupt and Error Handler Declarations with SICLCALLBACK

Custom error, SRQ, and interrupt handler (callback) functions installed using SICL's `ionerror`, `ionsrq`, and `ionintr` functions should be declared using the SICL modifier `SICLCALLBACK`, which is defined as "`_export _far _pascal`" in `sicl.h`. Failure to do this usually causes a "General Protection Fault" error at the time the handler is called.

Example declarations:

```
void SICLCALLBACK my_err_handler(INST id, int error) {
   /* your code here */
}

void SICLCALLBACK my_int_handler(INST id, int reason, long sec) {
   /* your code here */
}

void SICLCALLBACK my_srq_handler(INST id, int reason, long sec) {
   /* your code here */
}
```

Additionally, if you are developing an application using the QuickWin feature provided with Microsoft compilers and are installing a custom handler, you must also use the `_loadds` modifier with your handler declaration as follows:

Example declaration for QuickWin applications:

```
void SICLCALLBACK _loadds my_err_handler(INST id, int error) {
   /* your code here */
}
```

Application Cleanup

SICL has defined a special function, `_siclcleanup()`, to insure that Windows performs the necessary clean-up required when a SICL program completes execution. Each SICL application should call `_siclcleanup()` before exiting or posting a WM_QUIT message in order to release resources allocated for the application by the SICL library. Without this call, you may experience difficulty in executing your application, especially from within debuggers.

Note that the `I_ERROR_EXIT` handler calls `_siclcleanup()` automatically before it exits.

Avoiding nested I/O

In order to allow other Windows 3.1 applications to execute while a SICL application is running, SICL may temporarily suspend execution during a SICL call while waiting for a slow HP-IB transaction to complete. Without this feature your Windows system would be locked up until the transaction completes. However, because Windows is an event/message driven operating system, it is possible that the SICL application would receive a message instructing it to initiate another SICL call before the first one completes. This will result in a SICL error. Your program must be designed so that this situation does not occur. The second example at the end of this chapter shows one design method to prevent this Windows 3.1 problem.

Memory Models

We strongly recommend that you use the large memory model when designing applications that call SICL functions. This is because SICL requires all pointer parameters to be "far" pointers. Most SICL function prototypes in the `sicl.h` header file explicitly declare all pointer parameters to be far. However, there is no way to declare pointer types for functions that take a variable number of arguments (such as SICL's formatted I/O routines) and your compiler will not be able to properly check or cast types for these functions.

SICL Libraries

All applications that use SICL must link to the `SICL16.LIB` import library and to one additional import library that is compiler dependent. Selecting the proper library depends on the compiler you are using and whether you are calling SICL functions from another dynamic link library (DLL) or from an application program.

`MSAPP16.LIB` Link to this library if you are developing an application with a Microsoft compiler.

`BCAPP16.LIB` Link to this library if you are developing an application with a Borland compiler.

`MSDLL16.LIB` Link to this library if you are developing a DLL with a Microsoft compiler.

`BCDLL16.LIB` Link to this library if you are developing a DLL with a Borland compiler.

All SICL libraries are located in the "C" directory under the SICL base directory (e.g. `C:\SICL\C` if you installed SICL in the default location).

Compiling and Linking

The following is a summary of important compiler specific considerations for several of the popular Windows 3.1 compiler products. If you are using a different tool, remember to keep these considerations in mind.

For Microsoft compilers such as Microsoft C 7.0 or SDK compilers:

- Make sure the large memory model is selected using `/AL`.
- Be sure to compile with an option that adds prolog code for exported functions (`/GA` or `/Gsw` for applications, `/GD` for DLLs). This causes the applications data segment to be loaded correctly at the beginning of an exported function and is required for SICL error and interrupt handlers to work correctly.

- You may wish to add the SICL "C" directory (e.g. `C:\SICL\C`) to the Include file and Library file search paths. These are typically set using the `LIB` and `INCLUDE` environment variables in the `AUTOEXEC.BAT` file in the root directory. Otherwise, the library and include files and paths should be explicitly specified in the makefile.

For Microsoft Visual C++:

- To set the memory model do the following:

  1. Select `Options|Project`.
  2. Click on the `Compiler` button, then select `Memory Model` from the `Category` list.
  3. Click on the `Model` list arrow to display the model options, and select `Large`.
  4. Click on `OK` to close the Compiler dialog box.

- You may wish to add the SICL "C" directory (e.g. `C:\SICL\C`) to the Include file and Library file search paths. They are set under the `Options|Directories` menu selection. Otherwise, the library and include files and paths should be explicitly specified in the project file.

For Borland C (or Turbo C):

- Make sure large memory model is selected.

  From the Integrated Development Environment, do the following:

  1. Select `Options|Compiler|Code Generation`.
  2. Click on the checkbox next to `Large` in the Model box.
  3. Click on `OK` to close the dialog box.

  You can do this from the command line environment by specifying the `/ml` option to the compiler.

- The Borland C linker defaults to being case insensitive when resolving references. To link to the SICL libraries, you will need to tell the linker to be case sensitive.

  To do this from Borland's Integrated Environment:

  1. Select `Options|Linker Settings`.
  2. Click on the checkbox next to `Case-sensitive exports`.
  3. Click on `OK` to close the dialog box.

  You can do this from the command line environment by specifying the `/C` option to TLINK.

SICL Message Logging

While developing your SICL application or tracking down problems, you may wish to use the SICL **Message Viewer** utility. This utility provides a debug window to which SICL logs internal messages during application execution, including those logged by the `I_ERROR_EXIT` and `I_ERROR_NOEXIT` error handlers. The Message Viewer utility provides menu selections for saving the logged messages to a file, and to clear the message buffer.

To start the utility, double click on the Message Viewer icon in the HP SICL group. The utility must be started before execution of the SICL application. It will receive messages while minimized, however.

SICL and Windows NT

The SICL Header File

The SICL header file `sicl.h` should be included at the beginning of each file containing functions that call SICL. The header file contains function prototypes for all SICL functions plus definitions for all SICL constants and error codes. To include the SICL header, simply add the following line to your source file:

```
#include "sicl.h"
```

The SICL header file is located in the "C" directory under the SICL base directory (e.g. `C:\SICL\C` if you installed SICL in the default location). You may wish to add this directory path to the include file path used by your C programming language tools.

`itimeout`

Make sure that an `itimeout` value has been set for all SICL sessions in your program. Otherwise, when an instrument does not respond to a SICL read or write, SICL will wait indefinitely in the SICL kernel access routine preventing the application from being killed. If this occurs, you must re-boot Windows NT to recover.

Thread Support

SICL can be used in multi-threaded designs and SICL calls can be made from multiple threads. However, there are a few important points to remember:

- SICL error handlers (installed with `ionerror`) are per-process, not per-thread, but are called in the context of the thread that caused the

error to occur. Calling `ionerror` from one thread will overwrite any error handler presently installed by another thread.

- The `igeterrno` is per-thread, and returns the last SICL error that occurred in the current thread.
- You may wish to make use of the SICL session locking functions (`ilock` and `iunlock`) to help coordinate common instrument accesses from more than one thread.

SICL Libraries

All applications and DLLs that use SICL must link to the `SICL32.LIB` import library.

The SICL library is located in the "C" directory under the SICL base directory (e.g. `C:\SICL\C` if you installed SICL in the default location). You may wish to add this directory to the library file path used by your language tools.

Other Libraries

You should use the `$(cvarsdll)` when compiling your application, and use `$(guilibsdll)` for Windows applications, or `$(conlibsdll)` for console applications when linking your application. This is due to the fact that you need to use the DLL version of the run-time library, as the run-time libraries contain global variables that must be shared between your application and the SICL DLL. If you use the static version of the run-time libraries, these global variables will not be shared, and unpredictable results could occur. For example, if you use `isscanf` with the `%F` format, an application error will occur.

Compiling

The following is a summary of important compiler specific considerations for several Windows NT compiler products. If you are using a different tool, remember to keep these considerations in mind.

For Microsoft SDK compilers:

- You may wish to add the SICL "C" directory (e.g. `C:\SICL\C`) to the include file and library file search paths set by the `Include` and `Lib` environment variables. Otherwise, the library and include files and paths should be explicitly specified in the makefile.
- Include `$(cvarsdll)` on the compile line in your makefile.
- Include `$(conflagsdll)` or `$(guiflagsdll)` on the link line in your makefile for console or Windows applications respectively.

For Microsoft Visual C++:

- Select **Options | Project** from the menu. Click on the **Compiler** button then select **Preprocessor** from the category box. Add **WIN32** to the **Symbols and Macros to Define**. Next select **Runtime** from the category box and select **Multi-Threaded using DLL**. SICL requires these definitions for Windows NT. Click on **OK** and **OK** to close the dialog boxes.
- Select **Options | Project** from the menu. Click on the **Linker** button then select **Input** from the category box. Add **sicl32.lib** to the **Object / Library Modules**. Otherwise, you may add the library directly to your project file.
- You may wish to add the SICL "C" directory (e.g. **C:\SICL\C**) to the Include file and Library file search paths. They are set under the **Options|Directories** menu selection. Otherwise, the library and include files and paths should be explicitly specified in the source and project files.

SICL Message Logging

SICL logs internal messages as Windows NT events. This includes error messages logged by the **I_ERROR_EXIT** and **I_ERROR_NOEXIT** error handlers. While developing your SICL application or tracking down problems, you may wish to view these messages, and you can do so by starting the **Event Viewer** utility in the **Administrative Tools** group. Both System and Application messages can be logged to the event viewer from SICL. SICL messages are identified by **SICL LOG** or by the driver name **hp341i32**.

Oscilloscope Windows Application Example

This example programs an oscilloscope (such as and HP 54601), up-loads the measurement data, and instructs the scope to print its display to a ThinkJet printer. The program uses many SICL features, as well as illustrating some important Windows programming information for SICL. An overview of the program follows the sections on building the program for Windows 3.1 and Windows NT.

The Oscilloscope example files are located in the `C\SAMPLES\SCOPE` directory under the SICL base directory (e.g. `C:\SICL\C\SAMPLES\SCOPE` if you installed SICL in the default location). The directory contains the source program and a number of files to help you build the example with specific compilers. Here is a summary of the files provided:

`SCOPE.C`	Example program source file.
`SCOPE.H`	Example program header file.
`SCOPE.RC`	Example program resource file.
`SCOPE.DEF`	Example program module definitions file.
`SCOPE.ICO`	Example program icon file.
`MSCSCOPE.MAK`	Windows 3.1 makefile for Microsoft C and Microsoft SDK compilers.
`VCSCOPE.MAK`	Windows 3.1 project file for Microsoft Visual C++.
`QCSCOPE.MAK`	Windows 3.1 project file for Microsoft QuickC for Windows.
`BCSCOPE.MAK`	Windows 3.1 makefile for Borland C command line compilation.
`BCSCOPE.PRJ`	Windows 3.1 project file for Borland C Integrated Development Environment.

If you have purchased and installed SICL for Windows NT, the following file will also be available:

`MSCSCPNT.MAK`	Windows NT makefile for Microsoft SDK compilers.
`VCSCPNT.MAK`	Windows NT project file for Microsoft Visual C++.

Building the Program for Windows 3.1

The process for creating the project file for the example using Microsoft Visual C is described below in order to summarize many of the considerations discussed earlier. You may also load the makefile directly from the from the `SAMPLES\SCOPE` directory if you desire. If you are using another language tool, choose the corresponding project file or makefile from the `SAMPLES\SCOPE` directory. An overview of the program is provided at the end this chapter.

To compile and link the example program with Microsoft Visual C, follow these steps:

1. Select `Project|New` from the menu, and enter a name (with the path of your working directory) for the project in the dialog box. Also select `Windows Application` as the project type. Click on the `OK` button.
2. The `Edit` dialog box will now be displayed. Double click on the source file `SCOPE.C` to add it to the project. Also add `SICL16.LIB` and `MSAPP16.LIB` from the SICL C directory (e.g. `C:\SICL\C` if you installed SICL in the default location). Click on the `Close` button.
3. Select `Option|Directories` from the menu and add the SICL "C" directory (for example, `C:\SICL\C`, if you installed SICL in the default location) to the end of the paths listed in the `Include` edit box. Don't forget to add a semicolon before the SICL path. Click the `OK` button.
4. Select `Options|Project`. Click on the `Compiler` button, then select `Memory Model` from the `Category` list. Click on the `Model` list arrow to display the model options, and select `Large`. Click on `OK` to close the Compiler dialog box.
5. Now select `Project|Build` to build the application.

If there are no errors reported, you can execute the program by selecting **Project|Execute**. An application window will be opened. Several commands are available from the **Actions** menu, and any results or output will be printed in the program window. To end the program, select **File|Exit** from the program menu.

Building the Program for Windows NT

The process for creating the project file for the example using Microsoft Visual C is described below in order to summarize many of the considerations discussed earlier. You may also load the makefile directly from the **SAMPLES\SCOPE** directory if you desire. If you are using another language tool, choose the appropriate project file or makefile from the samples directory. An overview of the program is provided at the end this chapter.

To compile and link the example program with Microsoft Visual C, follow these steps:

1. Select **Project|New** from the menu, and enter a name (with the path of your working directory) for the project in the dialog box. Also select **Windows Application** as the project type. Click on the **OK** button.
2. The **Edit** dialog box will now be displayed. Double click on the source file **scope.c** to add it to the project. Also add **sicl32.lib** from the SICL C directory (e.g. **C:\SICL\C** if you installed SICL in the default location). Click on the **Close** button.
3. Select **Options | Project** from the menu. Click on the **Compiler** button then select **Preprocessor** from the category box. Add **WIN32** to the **Symbols and Macros to Define**. Next select **Runtime** from the category box. Select **Multi-Threaded using DLL**. Click on **OK** and **OK**.
4. Select **Options | Project** from the menu. Click on the **Linker** button then select **Input** from the category box. Add **sicl32.lib** to the **Object / Library Modules**. Otherwise, you may add the library directly to your project file.
5. Select **Option|Directories** from the menu and add the SICL "C" directory (for example, **C:\SICL\C**, if you installed SICL in the default location) to the end of the paths listed in the **Include** files edit box. Don't forget to add a semicolon before the SICL path. Click the **OK** button.

6. Now select **Project|Build** to build the application.

If there are no errors reported, you can execute the program by selecting **Project|Execute**. An application window will be opened. Several commands are available from the **Actions** menu, and any results or output will be printed in the program window. To end the program, select **File|Exit** from the program menu.

Program Overview

Because of the length of the example, the full program is not included here. You may want to view the program with an editor as you read through this section. The program is designed to illustrate individual SICL features and programming considerations, and is not meant to be a robust Windows application.

Be sure to refer to the HP SICL *Reference Manual* or SICL **Help** for more detailed information on the SICL features discussed below, and to learn about all of the functions provided by SICL.

Custom Error Handler

The Oscilloscope program defines a custom error handler that will be called whenever an error occurs during a SICL call. The handler is installed using **ionerror** before any other SICL function call is made, and will be used for all SICL sessions created in the program.

```
void SICLCALLBACK my_err_handler(INST id, int error)
{
    ...
    sprintf(text_buf[num_lines++],
            "session id=%d, error = %d:%s", id, error, igeterrstr(error));
    sprintf(text_buf[num_lines++], "Select 'File | Exit' to exit program!");
    ...

    // If error is from scope, disable I/O actions by graying out menu picks.
    if (id == scope) {
        ... code to disallow further I/O requests from user

    }
}
```

The error number is passed to the handler, and **igeterrstr** is used to translate the error number into a more useful description string. If desired, different actions can be taken depending on the particular error or id that caused the error.

Locks

SICL allows multiple applications to share the same interfaces and devices. Different applications may access different devices on the same interface, or may alternately access the same device (a shared resource). If your program will be executing along with other SICL applications, you may wish to prevent another application from accessing a particular interface or device during critical sections of your code. SICL provides the **ilock/iunlock** functions for this purpose.

```
void get_data (INST id)
{
    ...  non-SICL code

    // lock the device to prevent access from other applications
    ilock(scope);

    ...  SICL I/O code to program scope and get data

    // release the scope for use by others
    iunlock(scope);

    ...  non-SICL code
}
```

Lock the interface or device with **ilock** before critical sections of code, and release the resource with **iunlock** at the end of the critical section. Using **ilock** on a device session prevents any other device session from accessing the particular device. Using **ilock** on an interface session prevents any other session from accessing the interface and any device connected to the interface. See the description of **isetlockwait** in the *Reference Manual* to determine what actions can be taken when a SICL call in your code attempts to access a resource that is locked by another session.

Formatted I/O

SICL provides extensive formatted I/O functionality to help facilitate communication of I/O commands and data. The example program uses just a few of the capabilities of the **iprintf/iscanf/ipromptf** functions and their derivatives.

As we have seen in the example at the beginning of the chapter, **iprintf** is used to send commands. As with all of the formatted I/O functions, the data is actually buffered. In this call, the "\n" at the end of the format

```
iprintf(id,":waveform:preamble?\n");
```

causes the buffer to be flushed and the string to be output. If desired, several commands can be formatted before being sent, and then all output at once. The formatted I/O buffers are automatically flushed whenever the buffer fills (see **isetbuf**) or when an **iflush** call is made.

When reading data back from a device, the **iscanf** function is used. To read the preamble information from the scope, we use the format string

```
iscanf(id,"%,20f\n",pre);
```

"%,20f\n". This string expects to input 20 comma separated floating point numbers into the "pre" array.

To upload the scope waveform data, the string "%#wb\n" is used. The "wb" indicates that we are reading word-wide binary data. The "#" preceding

```
iscanf(id,"%#wb\n",&elements,readings);
```

the data modifier tells **iscanf** to get the maximum number of binary words to read from the next parameter (&elements). The read will continue until an EOI indicator is received, or the maximum number of words has been read.

Interface Sessions

Sometimes it may be necessary to control the HP-IB bus directly instead of using SICL commands that do it for you. This is accomplished using an interface session and interface specific commands. Here we use **igetintfsess** to get a session for the interface to which the scope we have been talking to is connected. (Since we know which interface is being used, it is also possible to just use an **iopen** call on that interface.) Then **igpibsendcmd** is used to send some specific command bytes out on the bus to tell the printer to listen and the scope to send its data. The **igpibatnctl** function directly controls the state of the ATN signal on the bus.

```
void print_disp (INST id)
{
    INST hpibintf ;
    ...

    hpibintf = igetintfsess(id);
    ...
```

```
    // tell scope to talk and printer to listen
    //    the listen command is formed by adding 32 to the device address
    //        of the device to be a listener
    //    the talk command is formed by adding 64 to the device address of
    //        the device to be a talker

    cmd[0] = (unsigned char)63 ;      // 63 is unlisten
    cmd[1] = (unsigned char)(32+1) ;  // printer at addr 1, make it a listener
    cmd[2] = (unsigned char)(64+7) ;  // scope at addr 7, make it a talker
    cmd[3] = '\0';                    // terminate the string

    length = strlen (cmd) ;

    igpibsendcmd(hpibintf,cmd,length);
    igpibatnctl(hpibintf,0);

    ...
}
```

SRQs and **iwaithdlr**

Many instruments are capable of using the service request, or SRQ, signal on the HP-IB bus to signal the controller that an event has occurred. If an application wishes to respond to SRQs, an SRQ handler must be installed with the **ionsrq** call. All SRQ handlers will be called whenever an SRQ occurs. In the example handler, the scope status is read to verify that the scope asserted SRQ, and then the SRQ is cleared and a status message is displayed. If the scope did not assert SRQ, the handler prints an error message.

```
void SICLCALLBACK my_srq_handler(INST id)
{
    unsigned char status;

    // make sure it was the scope requesting service
    ireadstb(id,&status);

    if (status &= 64) {
       // clear the status byte so the scope can assert SRQ again if needed.
       iprintf(id,"*CLS\n");

       sprintf(text_buf[num_lines++],
                "id = %d, SRQ received!, stat=0x%x", id,status);
    } else {
       sprintf(text_buf[num_lines++],
                "SRQ received, but not from the scope");
    }
    InvalidateRect(hWnd, NULL, TRUE);
}
```

In the routine that commands the scope to print its display, the scope is set to assert SRQ when printing is finished. While the scope is printing, we want the application to suspend execution. SICL provides the function **iwaithndlr** that will suspend execution and wait until either an event occurs that would call a handler, or a specified timeout value is reached. In the example, interrupt events are turned off with **iintroff** so that the application will respond only to SRQ events. Then the SRQ handler is installed with **ionsrq**. Code to program the scope to print and send an SRQ is next, then the call to **iwaithdlr**, with a timeout value of 30 seconds. When the scope finishes printing and sends the SRQ, the SRQ handler will be executed and then **iwaithdlr** will return. A call to **iintron** re-enables interrupt events.

```
void print_disp (INST id)
{
    ...

    iintroff();
    ionsrq(id,my_srq_handler);   // Not supported on HP 82335

    // tell the scope to SRQ on 'operation complete'
    iprintf(id,"*CLS\n");
    iprintf(id,"*SRE 32 ; *ESE 1\n") ;

    // tell the scope to print
    iprintf(id,":print ; *OPC\n") ;

    ... code to tell the scope to print

    // wait for SRQ before continuing program

    iwaithdlr(30000L);
    iintron();

    sprintf (text_buf[num_lines++],"Printing complete!") ;
    ...
}
```

Nested I/O and Windows 3.1

On Windows 3.1, programs must be designed so that a new SICL call cannot be initiated until the previous call completes. In this program, it would be possible for the user to select another action from the program menu that required a SICL call before the previous action completed. To prevent this, any action that uses SICL functions first disables all other actions by using Windows commands to gray out and disable other I/O menu items (see the

enable_io_menu_items function). These menu items are then re-enabled after the desired SICL calls have completed.

```
void print_disp (INST id)
{
    ... non-SICL code

    // do this before making all SICL calls
    enable_io_menu_items(FALSE);

    ... SICL I/O code

    // do this after making all SICL calls
    enable_io_menu_items(TRUE);
}
```

6

Troubleshooting

Troubleshooting

This chapter contains a description of SICL error codes and hints for tracking down common problems and resolving errors with SICL programs.

Error Codes

When you install a default SICL error handler such as **I_ERROR_EXIT** or **I_ERROR_NOEXIT** with an **ionerror** call, a SICL internal error message will be logged. To view these messages on Windows 3.1, you should start the **Message Viewer** utility by double clicking on the icon in the HP SICL group. You may wish to iconify the utility during execution of your program, but you must always start it up before you execute a program in order for messages to be logged.

On Windows NT, SICL logs internal messages as Windows NT events and you can view these messages by starting the **Event Viewer** utility in the **Administrative Tools** group. Both System and Application messages can be logged to the **Event Viewer** from SICL. SICL messages are identified by **SICL LOG**, or by **hp341i32** which is the HP-IB driver name.

You may also use **ionerror** to install your own custom error handler. Your error handler can call **igeterrstr** with the given error code and the corresponding error message string will be returned.

The following table contains an alphabetical summary of SICL error messages:

Error Messages

Name	Description
I_ERR_ABORTED	A SICL call was aborted by `iabort` or external means.
I_ERR_BADADDR	The device/interface address passed to iopen doesn't exist. Verify that the interface name is the one assigned using `I/O Config`.
I_ERR_BADCONFIG	An invalid configuration was identified.
I_ERR_BADFMT	Invalid format string specified for `iprintf` or `iscanf`.
I_ERR_BADID	The specified `INST id` is invalid (no corresponding iopen).
I_ERR_BUSY	Interface is in use by a non-SICL entity.
I_ERR_DATA	Data integrity violation (CRC, Checksum, etc.).
I_ERR_INVLADDR	Invalid address. The specified address is not a valid address (e.g. "hpib,57").
I_ERR_IO	Generic I/O error.
I_ERR_LOCKED	Resource is locked by another session (see `isetlockwait` intrinsic).
I_ERR_NESTEDIO	Attempt to call another SICL function when current SICL function has not completed (Windows 3.1).
I_ERR_NOCMDR	Commander is not active or available, or doesn't exist.
I_ERR_NOCONN	No connection.
I_ERR_NODEV	Device is not active.
I_ERR_NOERROR	No Error (Value is zero (0)).

Error Codes (cont.)

Name	Description
I_ERR_NOINTF	Interface is not active
I_ERR_NOLOCK	An unlock was specified when device wasn't locked.
I_ERR_NOPERM	Permission denied (access rights violated).
I_ERR_NOTIMPL	Call not supported on this implementation (for example, **ionsrq** is not supported with the HP 82335 interface).
I_ERR_NOTSUPP	Not a supported operation.
I_ERR_OS	SICL encountered an operating system error.
I_ERR_OVERFLOW	Arithmetic overflow.
I_ERR_PARAM	The constant or parameter passed is not valid for this call.
I_ERR_SYMNAME	Symbolic name passed to **iopen** not recognized.
I_ERR_SYNTAX	Syntax error occurred parsing address passed to **iopen**. Make sure that you have formatted the string properly. White space is not allowed.
I_ERR_TIMEOUT	A timeout occurred on the read/write operation. The device may be busy, in a bad state, or you may need a longer timeout value for that device. Check also that you passed the correct address to **iopen**.

Common SICL Problems with Windows 3.1

Unresolved SICL externals when building a SICL application

Check that you are linking to the correct import libraries. Refer to the "Libraries" discussion in the "Getting Started" chapter.

Can't find "llibxxxx" when building a SICL application

You probably did not load the large memory model libraries when you installed your compiler software. Re-run the setup program for you compiler and specifically request that large memory model libraries be installed.

Subsequent execution of SICL application gives strange behavior

Check that `_siclcleanup` is called at the end of the program and at any other possible program exit point (error handlers, etc.).

General Protection Fault occurs when interrupt, SRQ or error handler called

Check that the interrupt or error handler routine was declared with the `SICLCALLBACK` modifer. Also, make sure that compiler options to generate prolog code for exported functions were selected. If you are using the QuickWin feature of Microsoft compilers, you must also use the `_loadds` modifier in the handler declaration. Refer to "Programming Considerations" in the "Getting Started" chapter.

General Protection Fault when calling SICL formatted I/O routine

Verify that all pointer parameters passed to SICL formatted I/O routines are declared as `_far` or that the compiler large memory model option is selected.

I_ERR_NESTED_IO occurs

A subsequent SICL function call has been made before a previous call completed. In order to allow other Windows 3.1 applications to execute while a SICL application is running, SICL may temporarily suspend execution in the middle of a SICL call while waiting for a slow HP-IB transaction to complete. Without this feature your Windows system would be locked up until the transaction completes. However, because Windows is an event/message driven operating system, it is possible that the SICL application would receive a message instructing it to initiate another SICL call before the first one completes. This will result in a SICL error. Your program must be designed so that this situation does not occur. The second example at the end of the

"Getting Started" chapter shows one design method to prevent this Windows 3.1 problem.

Common SICL Problems with Windows NT

Program appears to hang and cannot be killed

Check that an `itimeout` value has been set for all SICL sessions in your program. Otherwise, when an instrument does not respond to a SICL read or write, SICL will wait indefinitely in the SICL kernel access routine preventing the application from being killed. You must re-boot Windows NT to recover.

Formatted I/O using %F causes application error

Verify that you are using `$(cvarsdll)` when compiling your application, and `$(guilibsdll)` for Windows applications, or `$(conlibsdll)` for console applications when linking your application.

A

HP-IB Fundamentals

HP-IB Fundamentals

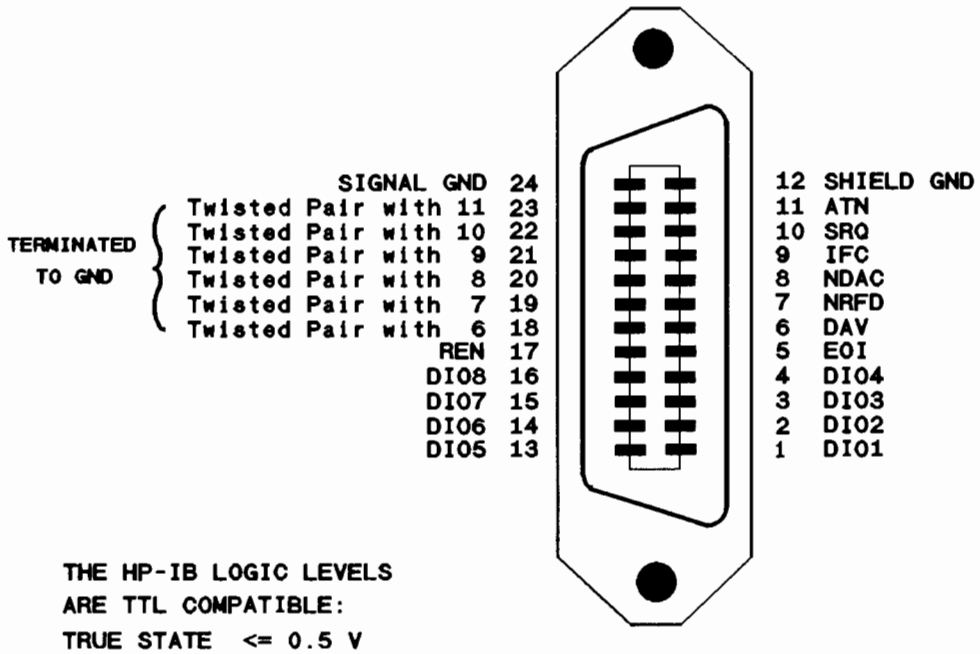
This appendix provides a summary of information contained in the IEEE-488 specification that defines the HP-IB bus. By using SICL device sessions, you can avoid the need to understand many of the bus intrinsics. But if you desire to program at the interface level or are doing extensive bus-level debugging, you may find this information helpful.

The following topics are discussed:

- HP-IB Description
- HP-IB Cabling
- HP-IB Abbreviations
- Commands and Data
- Controllers, Talkers, and Listeners
- Bus Commands
- Service Requests
- Table of ASCII Codes

The Hewlett-Packard Interface Bus (HP-IB) is HP's implementation of the IEEE-488 communication interface. It is used by a variety of instruments, disk drives, and peripherals manufactured by Hewlett-Packard and other companies. HP-IB is a 16-line bus that connects up to 15 devices in parallel on a communication link.

The following figure shows the HP-IB connector.



THE HP-IB LOGIC LEVELS
ARE TTL COMPATIBLE:
TRUE STATE <= 0.5 V
FALSE STATE >= 2.0 V

Of the 16 signal lines, 8 are data lines, 3 are for handshake purposes, and the remaining 5 are control lines. Information is transferred across the eight data lines in a bit-parallel, byte-serial fashion. Briefly, the eight control and handshake lines are used as follows:

ATN — Attention is used primarily to differentiate between Command mode and Data mode. When ATN is true, information on the data lines is interpreted as a bus command; when ATN is false, the information is treated as a data byte.

EOI — End Or Identify has two uses. EOI is asserted on the last byte of a data transfer—this signals all listening devices that no more data should be expected on the transfer. EOI is used in combination with ATN to perform a parallel poll.

IFC — Interface Clear is under the exclusive control of the system controller. When it is pulsed true, all device interfaces are returned to an idle state and the state of the bus is cleared.

REN — Remote Enable may be set by the system controller to permit devices to operate in Remote mode—that is, under programmed HP-IB control instead of via the device's front panel.

SRQ — Service Request can be set by a device on the interface to indicate it is in need of service. SRQ might be set at the completion of a task such as taking a measurement, when an error is detected during device operation, or when requesting to be active controller.

DAV — Data Valid is a handshake line indicating that the active talker has placed data on the data lines (DIO1 through DIO8).

NRFD — Not Ready For Data is a handshake line indicating that one or more active listeners is not ready for more data, and the active talker should wait before sending new data on the bus.

NDAC — Not Data Accepted is a handshake line indicating that one or more active listeners has not accepted the current data byte, and the active talker should leave the current byte asserted on the data lines.

HP-IB Cabling

An HP-IB system may be connected together in any configuration (star or linear or combination) as long as the following rules are followed:

- The total number of devices is less than or equal to 15.
- It is recommended that no more than three cable connector blocks be stacked on top of one another to minimize stress on connector mountings.
- The connector screws are designed to be tightened with fingers only. The screwdriver slots in the lock screws are for removal purposes only.
- For operation with data transfer rates less than 500KBytes/sec, the total length of the all cables used is less than or equal to 2 meters times the number of devices connected together up to a maximum of 20 meters. For example, the maximum cable length for 2 devices is 4 meters. The length between adjacent devices is not critical as long as the overall restriction is met. Systems should operate normally with up to one third of the devices powered off. Systems using older devices that do not use tri-state drivers will be limited to transfer rates of less than 250KBytes/sec.
- For operation with data transfer rates over 500KBytes/sec, the total length of the all cables used is less than or equal to 1 meter times the number of devices connected together up to a maximum of 15 meters. You should minimize cable length as much as possible. All devices must have tri-state drivers and must be powered on. Few instruments are capable of running at these speeds.
- Turning a device on or off while a system is running may cause faulty operation.

HP-IB bus extenders are also available that will allow operation over much greater distances.

There are two modes of communication on HP-IB: Command mode and Data mode.

In Command mode, information transmitted across the eight data lines is interpreted as talk or listen addresses, or universal address or unaddress commands (explained later). In this mode, only seven of the data lines are used. Some devices use the eighth line as a parity check for certain protocols.

In Data mode, any eight-bit value can be transmitted. The HP-IB can therefore be used for transmission of binary data as well as ASCII characters.

The three-line handshake scheme has several advantages. First, data transfer is asynchronous—the data rate is limited only by the speed of the devices actively involved in the transfer. A second, related advantage is that devices with different I/O speeds can be interconnected without need for other synchronization mechanisms. Also, multiple devices can be addressed concurrently.

Summary of HP-IB Command Abbreviations

The following list summarizes the standard low level HP-IB command abbreviations (mnemonics) in common use. They will be discussed in more detail in the following sections.

Mnemonic	Definition
ATN	Attention
DCL	Device Clear
EOI	End or Identify
EOL	End of Line
GET	Group Execute Trigger
GTL	Go To Local
IFC	Interface Clear
LAD	Listen Address
LLO	Local Lockout
MLA	My Listen Address
MTA	My Talk Address
OSA	Other Secondary Address
PPC	Parallel Poll Configure
PPD	Parallel Poll Disable
PPU	Parallel Poll Unconfigure
REN	Remote Enable
SDC	Selected Device Clear
SPD	Serial Poll Disable
SPE	Serial Poll Enable
SRQ	Service Request
TAD	Talk Address
UNL	Unlisten
UNT	Untalk

Controllers, Talkers, and Listeners

To understand communication among devices, you should be familiar with the concepts of controller, talker, and listener.

Controller

Two types of controllers are defined within an HP-IB system: system controller and active controller.

There must be a single *system controller* capable of taking control of the interface at any time. The system controller has exclusive control over the IFC and REN lines.

Each system also has one or more devices capable of being *active controller* (sometimes referred to as controller-in-charge), although there may be only one active controller at any given time. The active controller has the ability to perform tasks such as establishing listeners and talkers, sending bus commands, performing serial polls.

In most systems, a single computer will be both the system controller and the only active controller. Some non-system controller devices may request service indicating their desire to be active controller in order to perform some operation such as plotting data or directly accessing disk drives. The current active controller may "pass control" to a requesting device to make it the active controller. In other systems, a system controller may not be capable of operating as non-active controller, and therefore no pass-control capabilities will exist. Note that system controller capabilities may not be transferred.

An HP-IB system can be configured in one of three ways, and it affects the transfer of data as described:

- No controller. This mode of data transfer is limited to a direct transfer between one device manually set to talk only, and one or more devices manually set to listen only.

- Single controller. In this configuration, data transfer can be from controller to devices (Command or Data mode), from a device to controller (Data mode only), or from a device to other devices (Data mode only).
- Multiple controllers. This mode of data transfer is similar to that of a single controller, with the requirement that active controller status be passable from one controller to another. In this configuration, one controller must be designated as the system controller. This controller is the only one that can control the IFC and REN lines.

  Control is passed to another controller by addressing it as a talker and commanding it to "take control" (TCT). (Note that the HP-IB Command Library does not support more than one controller—therefore, control may not be passed.)

Talker

In each system there can be at most one device addressed as talker at any given time. A device becomes addressed as talker by receiving its talk address from the active controller. Each device on the bus must have a unique bus address. This address is usually set at the manufacturing site, but it may be set by switches on the instrument.

The addresses are in the range 0 to 30. A talk address is formed by adding the primary bus address to the talk address base value of 64 and transmitting that value across the data lines while ATN is asserted. For example, talk address 9 would be formed by asserting ATN and transmitting a byte whose value is 73 (64 + 9 = 73, ASCII character "I").

Listener

Listen addresses are formed in a similar manner to talk addresses, except that listen addresses use a base of 32. For example, listen address 9 is sent as value 41 transmitted with ATN true (32 + 9 = 41, ASCII character ")").

Multiple devices may be addressed to listen at any time, and data bytes will be received by all listeners in parallel. However, most devices cannot be addressed to both talk and listen at the same time. (See the table at the end of this appendix for talk and listen address codes.)

Extended Addressing

The descriptions of talk address and listen address refer to a device's primary address. Some devices also have extended talker or extended listener capabilities, sometimes used as secondary addresses or as device-dependent commands. With extended addressing, talk and listen addresses are represented by two command bytes. The first byte is the primary talk or listen address as described above. The second byte is a secondary address command.

Secondary addresses may be in the range 0 to 31. The secondary commands transmitted are formed by adding the secondary address to the base value 96 and transmitting the byte with ATN true.

Extended addresses can be used, for example, to access a specific I/O card within an instrument that allows multiple I/O cards such as a VXI cardcage. For example, if you are connecting to a VXI card cage through an HP E1405/6 Command Module or equivalent, the primary address passed to **iopen** corresponds to the address of the Command Module, and then a secondary address must be specified to select a specific instrument in the card cage. Secondary addresses of 0, 1, 2, ... 31, correspond to VXI instruments at logical addresses of 0, 8, 16, ... 248, respectively. For example, the **iopen** address string "hpib7,16,0" would specify the VXI instrument at logical address 0 (usually the Command Module), accessed through a Command Module at address 16 on the HP-IB bus.

Five types of information are transmitted when the bus is operating in Command mode (that is, when ATN is asserted):

- Talk addresses.
- Listen addresses.
- Universal commands.
- Addressed commands.
- Unaddress commands.

Talk addresses and listen addresses are discussed above. The other categories are described below.

Universal Commands

Universal commands are received by all responding devices on the bus whether addressed to listen or not. The commands are listed below.

Mnemonic	Command	Description
LLO	Local Lockout	Disables the front panel of the responding device. The REN line must be asserted in order for LLO to have any effect. If the instrument is already in Remote mode, the lockout will be immediate. Otherwise, the lockout will commence when the device receives its listen address.
DCL	Universal Device Clear	All devices capable of responding are returned to some known, device-dependent state. In some cases a device will perform a self-test in response to a Universal Device Clear.
PPU	Parallel Poll Unconfigure	Directs all devices on the HP-IB that have parallel poll configure capabilities to not respond to a parallel poll.
SPE	Serial Poll Enable	Enables Serial Poll mode on the interface.
SPD	Serial Poll Disable	Disables Serial Poll mode on the interface.

Addressed Commands

Addressed commands are executed only by those devices that are currently addressed as listeners. They allow the controller to initiate a simultaneous action by a selected group of devices on the bus, such as triggering them to take readings at the same time. The commands are listed below.

Mnemonic	Command	Description
SDC	Selected Device Clear	Similar to a Universal Device Clear (DCL) with only those devices addressed to listen responding.
GTL	Go To Local	Returns devices that are addressed to listen to Local mode (re-enables front panel programming). REN stays asserted when a GTL is sent, and devices will be returned to Remote upon receipt of their listen address.
GET	Group Execute Trigger	Initiates some preprogrammed action by listening devices. This may be used to simultaneously start action in a group of devices that are addressed to listen.
PPC	Parallel Poll Configure	Configures a device to respond to a parallel poll on a specified data line with either a positive or negative signal. A secondary command sent after PPC contains the data that configures the device.
TCT	Take Control	Transfers active controller status to another device on the bus. (This command is not permitted in the HP-IB Command Library.)

Unaddress Commands

The two unaddress commands can be considered as extensions of talk and listen addresses.

UNL (Unlisten) causes all devices on the bus (except those that have a built-in switch set to Listen Only) to stop being listeners. UNL is equivalent to listen address 31.

UNT (Untalk) directs any device on the interface to no longer be addressed as talker. Since there may only be one device addressed to talk at any time, receipt of another device's talk address is equivalent to receiving a UNT. UNT is equivalent to talk address 31.

Service Requests

Some devices that operate on the interface have the ability to request service from the system controller. A device may request service when it has completed a measurement, when it has detected a critical condition, or under many other circumstances.

A service request (SRQ) is initiated when the device sets the SRQ line true. The controller, sensing that SRQ has been set (typically either by polling the status of the line or by enabling an SRQ interrupt), can poll devices in one of two ways: serial poll or parallel poll.

Serial Poll

A typical sequence of events in performing a serial poll is:

- Establish a device as a talker.
- Send SPE to set up Serial Poll mode.
- Wait for the addressed device to send its serial poll response byte.
- Send an SPD and UNT to disable the Serial Poll mode.

The meaning of the serial poll response byte depends upon the individual device. However, if bit 6 of the response byte (bit value 64) is 1, the device is indicating it has requested service. If bit 6 is 0, the polled device is not the one that requested service. Individual device manuals provide additional information on the meanings of serial poll response bytes.

Parallel Poll

Parallel polling permits the status of multiple devices on HP-IB to be checked simultaneously. Each device is assigned a data line (DIO1 through DIO8) that the device sets true during the parallel poll routine if it requires service.

More than one device can be assigned to a particular data line. If a shared line is sensed true, a serial poll can typically be performed to determine which device set the line. A parallel poll is started when the controller asserts ATN and EOI together. After a short period of time the controller reads the poll byte and begins its interpretation thereof.

Some devices can be configured (by the PPC command) to respond on specific data lines. Other devices may respond on lines selected by switches or jumpers in the devices. Some devices do not have parallel poll capability.

ASCII Codes

The following table lists ASCII codes in decimal and hex, characters, and corresponding HP-IB commands. You may find these codes helpful for debugging, or for creating interface session commands.

Dec	Hex	Char	Cmd	Dec	Hex	Char	Cmd	Dec	Hex	Char	Cmd	Dec	Hex	Char	Cmd
0	00	NUL		32	20	SP	LO	64	40	@	TO	96	60	‘	
1	01	SOH	GTL	33	21	!	L1	65	41	A	T1	97	61	a	
2	02	STX		34	22	"	L2	66	42	B	T2	98	62	b	
3	03	ETX		35	23	#	L3	67	43	C	T3	99	63	c	
4	04	EOT	SDC	36	24	$	L4	68	44	D	T4	100	64	d	
5	05	ENQ	PPC	37	25	%	L5	69	45	E	T5	101	65	e	
6	06	ACK		38	26	&	L6	70	46	F	T6	102	66	f	
7	07	BEL		39	27	'	L7	71	47	G	T7	103	67	g	
8	08	BS	GET	40	28	(	L8	72	48	H	T8	104	68	h	
9	09	HT	TCT	41	29	)	L9	73	49	I	T9	105	69	i	
10	0A	LF		42	2A	*	L10	4A	74	J	T10	106	6A	j	
11	0B	VT		43	2B	+	L11	75	4B	K	T11	107	6B	k	
12	0C	FF		44	2C	,	L12	76	4C	L	T12	108	6C	l	
13	0D	CR		45	2D	–	L13	4D	77	M	T13	109	6D	m	
14	0E	SO		46	2E	.	L14	78	4E	N	T14	110	6E	n	
15	0F	SI		47	2F	/	L15	79	4F	O	T15	111	6F	o	
16	10	DLE		48	30	0	L16	80	50	P	T16	112	70	p	
17	11	DC1	LLO	49	31	1	L17	81	51	Q	T17	113	71	q	
18	12	DC2		50	32	2	L18	82	52	R	T18	114	72	r	
19	13	DC3		51	33	3	L19	83	53	S	T19	115	73	s	
20	14	OC4	DCL	52	34	4	L20	84	54	T	T20	116	74	t	
21	15	NAK	PPU	53	35	5	L21	85	55	U	T21	117	75	u	
22	16	SYN		54	36	6	L22	86	56	V	T22	118	76	v	
23	17	ETB		55	37	7	L23	87	57	W	T23	119	77	w	
24	18	CAN	SPE	56	38	8	L24	88	58	X	T24	120	78	x	
25	19	EM	SPD	57	39	9	L25	89	59	Y	T25	121	79	y	
26	1A	SUB		58	3A	:	L26	90	5A	Z	T26	122	7A	z	
27	1B	ESC		59	3B	;	L27	91	5B	[	T27	123	7B	{	
28	1C	FS		60	3C	<	L28	92	5C	\	T28	124	7C	|	
29	1D	GS		61	3D	=	L29	93	5D	]	T29	125	7D	}	
30	1E	RS		62	3E	>	L30	94	5E	^	T30	126	7E	~	
31	1F	US		63	3F	?	UNL	95	5F	_	UNT	127	7F	DEL	

B

Porting from the HP 82335 Command Library

Porting from the HP 82335 Command Library

The following table provides a cross reference between HP 82335 Command Library commands and SICL calls. It can be used as an aide for porting programs written for the older style calls to using SICL calls instead.

Remember to add `iopen`, `iclose`, and `_siclcleanup` calls to the program. Additionally, SICL provides other features (such as error handling) that you may wish to take advantage of instead of doing a straight translation of your existing code.

Definitions:

—n/a—	means 'Not Available'.
auto	means 'Happens Automatically' with no commands needed.

	82335 DOS Command Lib	82335 Windows DLL	SICL
SESSION CONTROL			
- Open/Close	auto	auto	iopen/iclose
- Lock/Unlock	—n/a—	—n/a—	ilock/iunlock
- Timeout	IOTIMEOUT	HpibTimeout	itimeout
- Error handler	—n/a—	—n/a—	ionerror
DATA OUTPUT			
- Unformatted	IOOUTPUTB	HpibOutputb	iwrite
- Formatted Strings	IOOUTPUTS	HpibOutputs	iprintf
- 488.2 Quoted Strings	—n/a—	—n/a—	iprintf
- ASCII Formatted Numbers			
- - Integer (NR1)	IOOUTPUT	HpibOutput	iprintf
- - Real number(NR2,NR3)	IOOUTPUT	HpibOutput	iprintf
- - Real array	IOOUTPUTA	HpibOutputa	iprintf
- Binary			
- - 16-bit Integers	IOOUTPUTB	HpibOutputb	iprintf
- - 32-bit Integers	IOOUTPUTB	HpibOutputb	iprintf
- - 32-bit Reals	IOOUTPUTB	HpibOutputb	iprintf
- - 64-bit Reals	IOOUTPUTB	HpibOutputb	iprintf
- 488.2 Binary			
- - #B, #Q, #H	—n/a—	—n/a—	iprintf
- - Arbitrary Block	IOOUTPUTAB	HpibOutputab	iprintf
- File Output	IOOUTPUTF	HpibOutputf	iprintf
DATA INPUT			
- Unformatted	IOENTERB	HpibEnterb	iread
- Formatted Strings	IOENTERS	HpibEnters	iscanf
- 488.2 Quoted Strings	—n/a—	—n/a—	iscanf
- ASCII Formatted Numbers			
- - Integer (NR1)	IOENTER	HpibEnter	iscanf
- - Real number(NR2,NR3)	IOENTER	HpibEnter	iscanf
- - Real array	IOENTERA	HpibEntera	iscanf

	82335 DOS Command Lib	82335 Windows DLL	SICL
DATA INPUT (cont'd)			
- Binary			
- -16-bit Integers	IOENTERB	HpibEnterb	iscanf
- -32-bit Integers	IOENTERB	HpibEnterb	iscanf
- -32-bit Reals	IOENTERB	HpibEnterb	iscanf
- -64-bit Reals	IOENTERB	HpibEnterb	iscanf
- 488.2 Binary			
- - #B, #Q, #H	—n/a—	—n/a—	iscanf
- - Arbitrary Block	IOENTERAB	HpibEnterab	iscanf
- File Input	IOENTERF	HpibEnterF	iscanf
DATA I/O CONTROL			
- Prompted Input	—n/a—	—n/a—	ipromptf
- Auto Byte Swap	YES	YES	YES
- DMA ON/OFF	IODMA	—n/a—	ihint
- EOI ON/OFF	IOEOI	HpibEoi	auto
- EOL string (length)	IOEOL	HpibEol	see note (2)
- Short T1 Capability	IOFASTOUT	HpibFastout	auto
- Set Match Char	IOMATCH	HpibMatch	itermchar
- Read termination reason	IOGETTERM	HpibGetterm	auto
INSTRUMENT CONTROL			
- CLEAR an instrument	IOCLEAR	HpibClear	iclear
- Put device in LOCAL	IOLOCAL	HpibLocal	ilocal or igpibrenctl
- Put device in REMOTE	IOREMOTE	HpibRemote	iremote or igpibrenctl
- TRIGGER a device	IOTRIGGER	HpibTrigger	itrigger
- Read stb (serial poll)	IOSPOLL	HpibSpoll	ireadstb

	82335 DOS Command Lib	82335 Windows DLL	SICL
BUS CONTROL			
- Send IFC	IOABORT	Hpibabort	iclaar
- Make all dev's REMOTE	IOREMOTE	HpibRemote	igpibrenctl
- Make all dev's LOCAL	IOLOCAL	HpibLocal	igpibrenctl
- CLEAR all devices	IOCLEAR	HpibClear	iclear
- Lock device front panel	IOLLOCKOUT	HpibLlockout	igpibllo
- Send bus commands	IOSEND	HpibSend	igpibsandcmd
- TRIGGER bus	IOTRIGGER	HpibTrigger	itrigger
- Reset to Power up state	IORESET	HpibReset	auto
- Set interface bus addr.	IOCONTROL	HpibControl	igpibbusaddr
- Get interface status	IOSTATUS	HpibStatus	igpibbusstatus
- Set/Drop ATN	IOCONTROL	HpibControl	igpibatnctl
PARALLEL POLL			
- Configure Parallel Poll	IOPPOLLC	HpibPpollc	igpibppollconfig
- Unconfigure parallel poll	IOPPOLLU	HpibPpollu	igpibppollconfig
- Conduct a parallel poll	IOPPOLL	HpibPpoll	igpibppoll
MISC. CAPABILITIES			
- SRQ Support	IOPEN	—n/a—	ionsrq
- Interrupt support	—n/a—	—n/a—	ionintr/isetintr
- Wait for event	—n/a—(1)	—n/a—	iwaithdlr
NON-CONTROLLER			
- Respond to serial poll	IOREQUEST	HpibRequest	isetstb
- Respond to parallel poll	—n/a—	—n/a—	igpibppollresp
- Request Service	IOREQUEST	HpibRequest	isetstb
- Pass Control	IOPASSCTRL	HpibPassctl	igpibpassctl
- Take Control	IOTAKECTRL	HpibTakectl	auto
- Can be non-Sys Ctlr?	YES	YES	YES

Notes:

1. Can be done manually using IOSTATUS in a loop.
2. The LF^END EOL sequence (required by 488.2) can be added with iprintf.

C

Interface Hardware Specifications

Interface Hardware Specifications

Radio and Television Interference

This device has been verified to comply with FCC Rules Part 15. Operation is subject to these two conditions: (1) this device may not cause radio interference, and (2) this device must accept any interference received (including interference that may cause undesired operation).

This equipment generates and uses radio frequency energy. If not installed and used in accordance with this manual, it can cause interference to radio and television communications. The rules with which it must comply afford reasonable protection against such interference when it is used in most locations. However, there can be no guarantee that such interference will not occur in a particular installation. If you think your computer is causing interference, turn off the system. If the radio or television reception does not improve, your computer is probably not causing the interference. If your computer does cause interference to radio and television reception, you are encouraged to try to correct the interference by one or more of the following measures:

- Relocate the radio or TV antenna.
- Move the computer away from the radio or television.
- Plug the computer into a different electrical outlet, so that the computer and the radio or television are on separate electrical circuits.
- Make sure you use only shielded cables to connect peripherals to your computer.
- Consult your dealer, Hewlett-Packard, or an experienced radio/television technician for other suggestions.
- Order the FCC booklet *How to Identify and Resolve Radio-TV Interference Problems* from the U.S. Government Printing Office, Washington, D.C. 20402. The stock number of this booklet is 004-000-00345-4.

Environmental Specifications

Specification	Minimum	Maximum
Power Supply Voltage	+4.75 volts	+5.25 volts
Power Supply Current	1.3 amperes typical (82341) 0.5 amperes typical (82335)	
Operating Temperature	0° C	+55° C
Non-Operating Temperature	-40° C	+70° C
Operating Humidity	15% RH	95% RH
Non-Operating Humidity		90% RH/24 hours

D

SICL/HP-IB System Information

SICL/HP-IB System Information

This chapter provides information on SICL software files and system interaction. This information can be used as a reference for removing SICL from the system if necessary.

Windows 3.1

File location

All SICL files are installed in the base directory specified by the user with the exception of several common files that Windows must be able to locate. These common files, `SICL16.DLL`, `SICLUT16.DLL`, and `HPIB.DLL`, are placed in the Windows directory.

Use of `WIN.INI`

SICL modifies the `WIN.INI` file in the Windows directory during installation. A `[SICL]` section is added and a `BASEDIR` declaration is made to record the location of the SICL directory on the system. This has no other impact on your Windows configuration.

SICL Configuration database

The `I/O Config` utility creates and maintains a `SICL.INI` file containing all SICL configuration information. This file is located under the SICL base directory. Under normal circumstances, this file should only be modified using the `I/O Config` utility.

Windows NT

File location

All SICL files are installed in the base directory specified by the user with the exception of several common files that Windows must be able to locate. These common files, `SICL32.DLL` and `HP341I32.SYS`, are placed in the `SYSTEM32` and `SYSTEM32\DRIVERS` directories under the main Windows directory.

The Registry

SICL places the following keys in the Windows NT registry under `HKEY_LOCAL_MACHINE`:

- Software\HP\SICL\CurrentVersion
- System\CurrentControlSet\Control\GroupOrderList
- System\CurrentControlSet\Control\ServiceOrderList
- System\CurrentControlSet\Services\hp341i\hpib
- System\CurrentControlSet\Services\EventLog\Application\SICLLog
- System\CurrentControlSet\Services\EventLog\System\hp341i32

SICL Configuration database

The `I/O Config` utility creates and maintains a `SICL.INI` file containing SICL configuration information. This file is located under the SICL base

directory. Under normal circumstances, this file should only be modified using the **I/O Config** utility.

Glossary

Glossary

address
A string uniquely identifying a particular interface or a device on that interface.

commander session
A session that communicates to the controller of this system.

controller
A computer used to communicate with a remote device such as an instrument. In the communications between the controller and the device the controller is in charge of, and controls the flow of communication (i.e. does the addressing and/or other bus management).

controller role
A computer acting as a controller communicating with a device.

device
A unit that receives commands from a controller. Typically a device is an instrument but could also be a computer acting in a non-controller role, or another peripheral such as a printer or plotter.

device driver
A segment of software code that communicates with a device. It may either communicate directly with a device by reading and writing registers, or it may communicate through an interface driver.

device session
A session that communicates as a controller specifically with a single device, such as an instrument.

handler
A software routine used to respond to an asynchronous event such as an error or an interrupt.

instrument
A device that accepts commands and performs a test or measurement function.

interface
A connection and communication media between devices and controllers, including mechanical, electrical, and protocol connections.

interface driver
A software segment that communicates with an interface. It also handles commands used to perform communications on an interface.

interface session
A session that communicates and controls parameters affecting an entire interface.

interrupt
An asynchronous event requiring attention out of the normal flow of control of a program.

lock
A state that prohibits other users from accessing a resource, such as a device or interface.

logical unit
A logical unit is a number associated an interface. A logical unit, in SICL, uniquely identifies an interface. Each interface on the controller must have a unique logical unit.

non-controller role
A computer acting as a device communicating with a controller.

process
An operating system object containing one or more threads of execution that share a data space. A multi-process system is a computer system that allows multiple programs to execute simultaneously, each in a separate process environment. A single-process system is a computer system that allows only a single program to execute at a given point in time.

session
An instance of a communications channel with a device. A session is established when the channel is opened with the `iopen` function and is closed with a corresponding call to `iclose`.

SRQ
Service Request. An asynchronous request (an interrupt) from a remote device indicating that the device requires servicing.

status byte
A byte of information returned from a remote device showing the current state and status of the device.

symbolic name
A name corresponding to a single interface. This name uniquely identifies the interface on this controller. If there is more than one interface on the controller, each interface must have a unique symbolic name.

thread
An operating system object that consists of a flow of control within a process. A single process may have multiple threads that each have access to the same data space within the process. However, each thread has its own stack and all threads may execute concurrently with each other (either on multiple processors, or by time-sharing a single processor).

Index

E2094-90001